Homework Assignment 3

ECE438

Due 5/12/2020

1. **(15 points)** For the following problem, assume a 5-stage pipelined processor with forwarding and hardware interlocking. Also, assume branch resolution is in the Execute stage. Consider the code below:

```
    addi $t2, $t1, 60
loop:
    lw $t4, 0($t1)
    lw $t5, 4($t1)
    xor $t6, $t4, $t5
    sw $t6, 8($t1)
    addi $t1, $t1, 12
    bne $t1, $t2, loop
```

   (a) How many loop iterations does the above code execute?

   This loop iterates 5 times because the exit condition is $t2 == t1$, $t2 = t1 + 60$, and $t2$ is incremented by 12 in each iteration.

   (b) Identify the data dependencies in the above code.

   - Between first `addi` and `bne` with $t2$
   - Between first `lw` and `xor` with $t4$
   - Between second `lw` and `xor` with $t5$
   - Between `xor` and `sw` with $t6$
   - Between second `addi` and `bne` with $t1$

   (c) Draw the pipeline execution diagram for the first *two* iterations of the above code when an "assume not taken" branching scheme <u>without</u> a branch delay slot is used.

clock cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`addi $t2, $t1, 60`	F	D	E	M	W																	
`lw $t4, 0($t1)`		F	D	E	M	W																
`lw $t5, 4($t1)`			F	D	E	M	W															
`xor $t6, $t4, $t5`				F	D	D	E	M	W													
`sw $t6, 8($t1)`					F	F	D	E	M	W												
`addi $t1, $t1, 12`							F	D	E	M	W											
`bne $t1, $t2, loop`								F	D	E	M	W										
`addi $t2, $t1, 60`									X	X	F	D	E	M	W							
`lw $t4, 0($t1)`												F	D	E	M	W						
`lw $t5, 4($t1)`													F	D	E	M	W					
`xor $t6, $t4, $t5`														F	D	D	E	M	W			
`sw $t6, 8($t1)`															F	~~D~~ F	D	E	M	W		
`addi $t1, $t1, 12`																~~F~~	~~D~~ F	D	E	M	W	
`bne $t1, $t2, loop`																	~~F~~	~~D~~ F	D	E	M	W

oops! I made a mistake here! hopefully my corrections make sense. copy & paste error !!

(d) How many clock cycles are required to execute the above code to completion when an "assume not taken" branching scheme <u>without</u> a branch delay slot is used?

The short answer: $1 + 9 * 4 + 7 + 4 = 48$ clock cycles
The long answer: First note that the problem asks how many cycles to execute the code to *completion*. We have one cycle for the first `addi`. Then we have four loop iterations in which the `bne` is "Taken," each requiring nine cycles (see diagram above). Then we have the last loop iteration in which the `bne` is "Not Taken," requiring seven cycles (no flushing). Finally, we have four cycles to drain the pipeline, i.e. to finish the `bne`.

(e) Modify the code to take advantage of a branch delay slot. How many clock cycles are required to executed your modified code to completion when an "assume not taken" branching scheme with a branch delay slot is used?

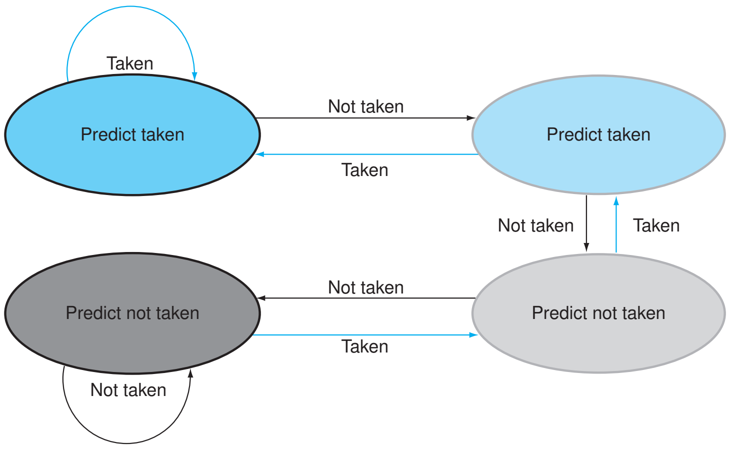
We can move the `sw` into the delay slot because it does not affect the `bne`. We have to remember to change the offset though because we are moving it after the `addi` that increments the base register, $t1$.

```
    addi $t2, $t1, 60
loop:
    lw $t4, 0($t1)
    lw $t5, 4($t1)
    xor $t6, $t4, $t5
    addi $t1, $t1, 12
    bne $t1, $t2, loop
    sw $t6, -4($t1)
```

In this case, we eliminate one of the two flushes for a "Taken" branch. Thus, the number of clock cycles to execute to completion with a branch delay slot is $1 + 8 * 4 + 7 + 4 = 44$

(f) How many clock cycles are required to execute you modified code assuming a 100% correct branch predictor in the decode stage in addition to the branch delay slot.

A 100% correct branch predictor means no flushes! Thus, the number of clock cycles required is $1 + 7 * 5 + 4 = 40$



(g) Consider the use of the two-bit predictor shown above in the decode stage. Assuming the predictor starts in the top right state, how many clock cycles are required to execute your code?

If the predictor is in the top right state (weakly Taken) when entering the loop, it will predict taken for the first execution of `bne` which will move the predictor state to the top left state (strongly Take). It will remain there until the execution of the last `bne`, in which case it will move back to the weakly Taken state. Thus, the predictor will predict taken for <u>all</u> executions of the `bne`. The number of required cycles is $1 + 7 * 5 + 1 + 4 = 41$ such that there is an extra cycle introduced due to misprediction during the last iteration of the loop.

(h) What is the accuracy of this predictor given the code above? Compare that to the accuracy of the "assume not taken" scheme.

In the case of our loop, we predict correctly 4 out of 5 times or $4/5 = 80\%$. Contrast that with the "assume not take" scheme which correctly predicts 1 out of 5 times or $1/5 = 20\%$.

(i) What speedup does the branch predictor from Problem 1g provide over the "assume not taken" scheme from Problem 1d?

Speedup $= 48/41 = 1.171$

(j) Further rearrange the code to reduce the number of stalls due to data dependencies.

We can simply move the second `addi` between the second `lw` and the `xor` to fill the load delay. This eliminates a stall per iteration. With our branch predictor and the branch delay slot, we can execute the code to completion now in $1 + 5 * 6 + 1 + 4 = 36$ clock cycles.

```
    addi $t2, $t1, 60
loop:
    lw $t4, 0($t1)
    lw $t5, 4($t1)
    addi $t1, $t1, 12
    xor $t6, $t4, $t5
    bne $t1, $t2, loop
    sw $t6, -4($t1)
```

2. **(30 points)** The following problems are related to techniques and terminology of the memory hierarchy.

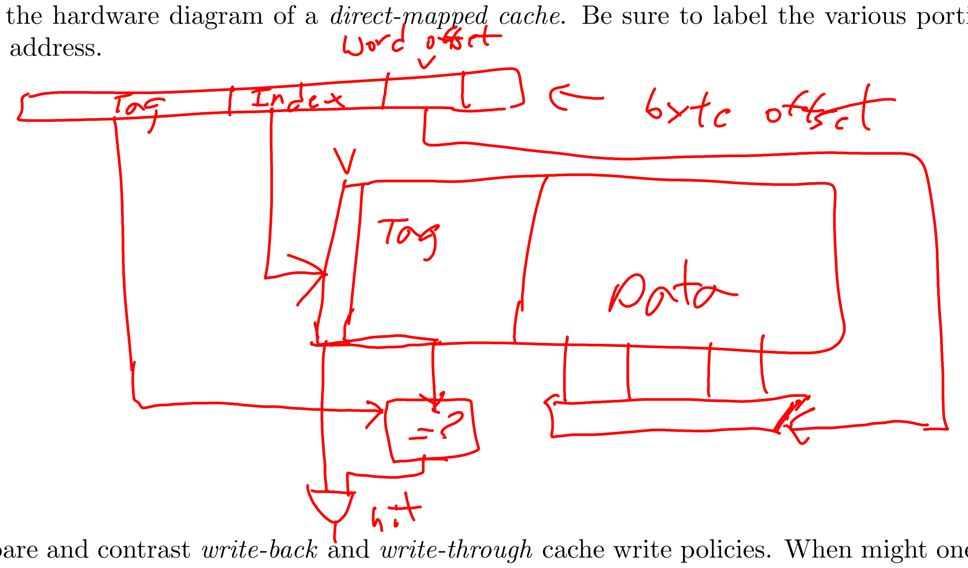
(a) Define spacial and temporal locality. Describe the role spacial and temporal locality play in the memory hierarchy.

The cache within the memory hierarchy rely on spacial and temporal locality. Spacial locality means that if a memory location is being accessed by the processor, there is a high likelihood that the nearby locations will be accessed as well. Thus, the cache will pull in an entire block of data. Temporal locality means that if a memory location is accessed, there is a high likelihood that it will be accessed again in the near future. Thus, caches hold recently used data for fast access in the event the data is used again.

(b) Define and briefly describe the types of cache misses talked about in lecture. Remember the three C's.

Compulsory: These are the initial misses due to an empty (cold) cache.
Conflict: These are misses due to a rigid block placement strategy, i.e. low associativity.
Capacity: These are misses due to the cache being too small to hold the entire working set of data and instructions.
Bonus round – Coherency: These are misses due to the cache coherence protocol used for sharing memory amongst processors.

(c) Draw the hardware diagram of a *direct-mapped cache*. Be sure to label the various portions of the address.



(d) Compare and contrast *write-back* and *write-through* cache write policies. When might one be preferred over the other?

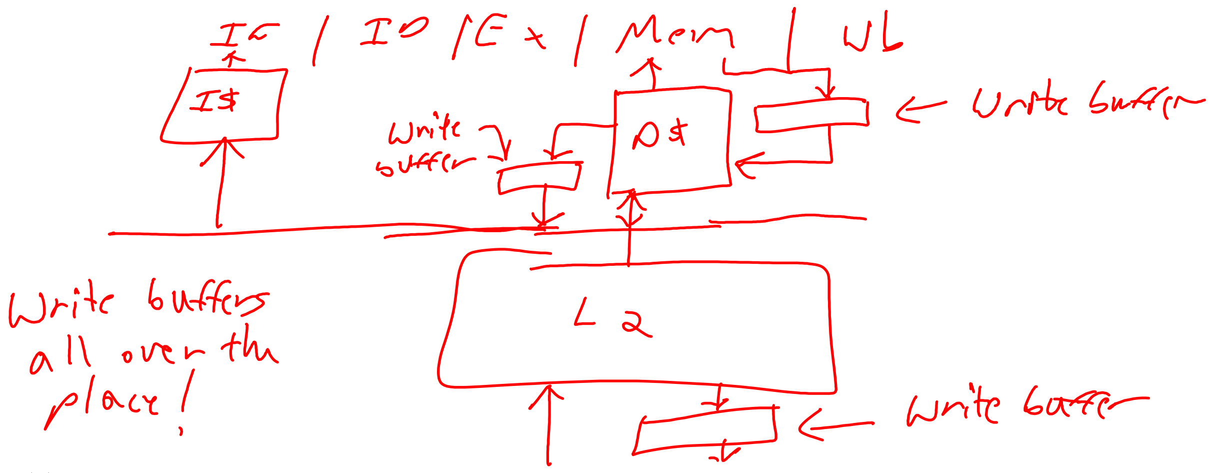
Write-back only writes data back to the next level of the memory hierarchy when a dirty block is being evicted, whereas write-through updates the next level of the memory hierarchy on every write. Write-back adds a little more complication to our cache but reduces traffic to the next level of the hierarchy. Write-back is almost always preferred; however write-through might be preferred in the case of a shared level of cache as the constant updates keep the memory consistent.

(e) Compare and contrast *write-allocate* and *write-no-allocate*. When might one be preferred over the other? Can either of these schemes be used with *write-back* and *write-through*? Why or why not?

Write-allocate vs. write-no-allocate policies address the issue of what to do in the case of a write to a block that is *not* in the cache. Write-allocate will act like a read miss and bring the block into the cache. Then it will modify the cache block. Write-no-allocate will simply forward the write to the next level if the given block is not currently in the cache. Write-no-allocate is advantageous in the case of a stream of writes such that the data is never read. In such case, precious cache space is not wasted on data that will never be read. Either of these schemes may be used for write-back or write-through but write-allocate makes little sense for a write-through cache, unless the cache is designed to not stall the processor while the given block is being brought in from the next level.

(f) Describe the purpose of a *write buffer*. Where in the memory hierarchy might you place a write buffer? Draw a diagram showing where you might place one and explain your reasoning.

A write buffer is used to buffer writes to the next level of the memory hierarchy to avoid stalling the processor. You can place a write buffer anywhere in the memory hierarchy. With a write-through L1, a write buffer allows instructions to continue flowing through the pipeline when store instructions are encountered. With a write-back cache, a write buffer allows the cache to prioritize reading the next level of memory.

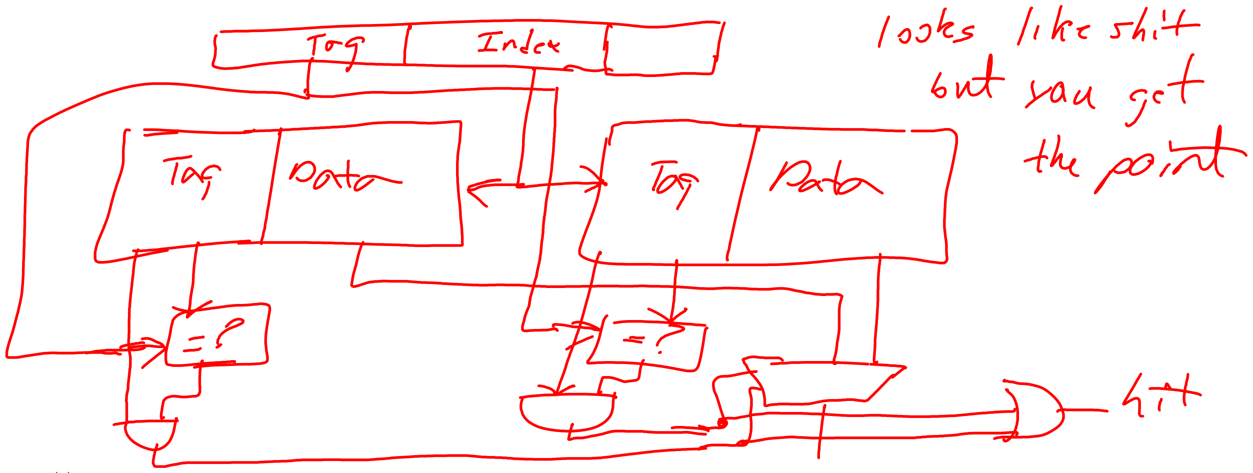


(g) What is the purpose of increasing the block size of a cache? What happens if the block size is too large?

Increasing block size allows the cache to take advantage of spacial locality and bandwidth. It's more efficient to bring in large amounts of data at once. If the block size is too large, however, the number of conflict misses will increase, causing an increase in the overall miss rate.

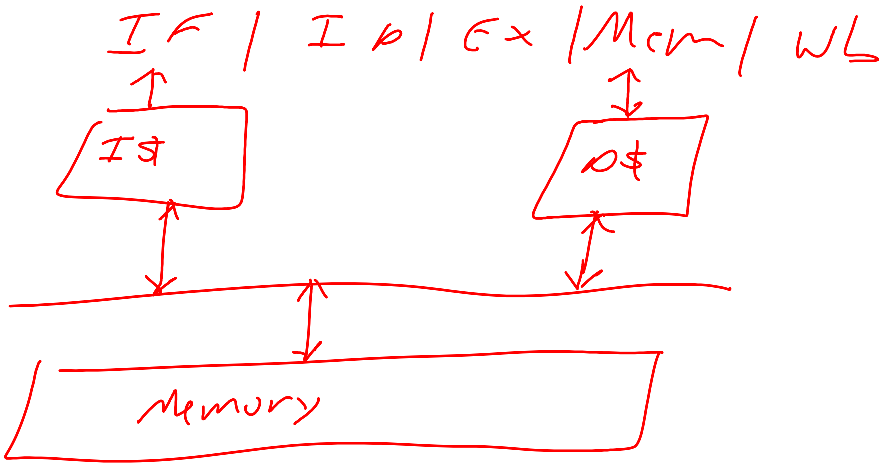
(h) Why might one add *associativity* to a cache? How might one add associativity? Draw a hardware diagram showing how to construct a 2-way set associative cache. What are the drawbacks to set-associative caches?

Adding associativity reduces conflict misses by providing a more flexible block placement strategy. The drawbacks to associativity include increased complexity, access (hit) time, and power as well as added difficult when pipelining the cache.



(i) What is a *split cache* scheme? Why is it beneficial? What might be a disadvantage? Draw a diagram of a single-level, split cache scheme.

A split cache system has a separate cache for instructions and data. The advantage of a split cache is it eliminates a structural hazard in the pipeline due to the fact that instruction and data access must happen simultaneously, and it allows each cache to be tuned specifically for the two different access patterns. Remember instruction access exhibits more locality. An advantage to the unified cache is it can dynamically adjusts the amount of cache allocated to instructions and data, where the split cache rigidly allocates based on the size of caches. Another disadvantage to the split cache system is overall system complexity, requiring the design of two separate caches and a mechanism to manage consistency between the two caches.



3. **(15 points)** For the problem below, a direct-mapped cache is provided the following sequence of *word* addresses: 3, 4, 5, 6, 7, 1, 1, 3, 36, 39, 35, 36

   (a) Assuming an initially empty, direct-mapped cache with 16 one-word blocks, complete the table below, identifying the tag and index for each word address. Also, indicate whether the access was a hit or miss, and if a it was a miss indicate the type of miss (i.e. compulsory or conflict).

Reference	**Address**	**Tag**	**Index**	**Hit/Miss**	**Type of Miss**
3	00000011	0000	0011	Miss	Compulsory
4	00000100	0000	0100	Miss	Compulsory
5	00000101	0000	0101	Miss	Compulsory
6	00000110	0000	0110	Miss	Compulsory
7	00000111	0000	0111	Miss	Compulsory
1	00000001	0000	0001	Miss	Compulsory
1	00000001	0000	0001	Hit	
3	00000011	0000	0011	Hit	
36	00100100	0010	0100	Miss	Conflict
39	00100111	0010	0111	Miss	Conflict
35	00100011	0010	0011	Miss	Conflict
36	00100100	0010	0100	Miss	Conflict

(b) Calculate the miss rate of the cache above. What is the hit rate?//

The miss rate is $10/12 = 83.3\%$. The hit rate is $2/12 = 16.6\%$

(c) Complete the table below assuming a direct-mapped cache with 8 two-word blocks.

Reference	Address	Tag	Index	Hit/Miss	Type of Miss
3	00000011	0000	001	Miss	Compulsory
4	00000100	0000	010	Miss	Compulsory
5	00000101	0000	010	Hit	
6	00000110	0000	011	Miss	Compulsory
7	00000111	0000	011	Hit	
1	00000001	0000	000	Miss	Compulsory
1	00000001	0000	000	Hit	
3	00000011	0000	001	Hit	
36	00100100	0010	010	Miss	Conflict
39	00100111	0010	011	Miss	Conflict
35	00100011	0010	001	Miss	Conflict
36	00100100	0010	010	Hit	

(d) Calculate the miss rate of the cache above. What is the hit rate?

The miss rate is $7/12 = 58.3\%$. The hit rate is $5/12 = 41.6\%$

(e) Compare the miss rate in part (d) with that of part (b). Does it improve? If so, why?

The miss rate improved by 25% with the larger block size by taking advantage of spacial locality!

(f) What two principles make memory caching effective? Be specific in your answer and provide a brief explanation of each principle.

Caches are effective because of spacial and temporal locality. Spacial locality is the observation that if a memory location is accessed, there is a high likelihood nearby locations will also be accessed, while temporal locality is the observation that if a memory location is accesses, there is a high likelihood it will be access again in the near future.

4. **(10 points)** Assuming a direct-mapped cache with a byte-address that is broken up such that bits 0-2 are for the byte offset, bits 3-7 are for the word offset, bits 8-14 are for the index, and bits 15-31 are for the tag, answer the following questions:

(a) How large are the words in this machine?

The words in this machine are 64 bits because 3 bits are used for the byte offset and $2^3 = 8$ bytes = 64 bits.

(b) How many words are in each cache block?

5 bits are used for the word offset, thus there are $2^5 = 32$ words in each cache block.

How many bytes are in each cache block?

$8 * 32 = 256$ bytes

(c) How many cache blocks are in the cache? How many sets are in the cache?

The index is 7 bits so $2^7 = 128$ sets in the cache, and because it's direct-mapped, there are also 128 blocks.

(d) How large is the data store of this cache?

The data store is $128 * 256 = 32$KiB

(e) How large is the tag store? Assume a *valid* and *dirty* bit are included with each tag.

Each tag is 19 bits and there are 128 tags so $19 * 128 = 2432$ bits

(f) If you were to modify the cache to be 2-way set associative but keep the data store the same size, what size would the tag and index be? How large would the tag store be?

The index would decrease by a bit while the tag would increase by a bit. Thus, the index would be 6 bits and the tag would be 20 bits (including valid and dirty). The number of tags is the same because each block needs a tag. The tag store would be $20 * 128 = 2560$ bits.

(g) How much memory can the above machine address?

The address is 32 bits in length so this machine can address 4GiB

5. **(10 points)** Imagine you have a 1GHz RISC processor with split L1 instruction and data caches, each 32kB in size with a hit time of 1ns. Access to main memory takes 30 ns, and 38% of instructions access memory. The L1 instruction cache miss rate is 0.7%, while the L1 data cache miss rate is 6%.

   (a) Calculate the Average Memory Access Time (AMAT) for each of the L1 caches.

   $AMAT_{I\$} = 1ns + 0.007 * 30ns = 1.21ns$
   $AMAT_{D\$} = 1ns + 0.06 * 30ns = 2.80ns$

   (b) Assuming your processor has a CPI of 1.2 with an ideal memory hierarchy, what is the CPI considering memory stalls?

   $CPI = 1.2cc/instr + 0.007 * 30cc/instr + 0.06 * 0.38 * 30cc/instr = 2.094cc/instr$

   (c) You are considering the inclusion of a 128kB L2 cache to improve your performance. If the miss rate of the L2 is 30%, what would the AMAT of the L1 instruction and data caches be with the L2 cache? Assume a 5ns L2 hit time.

   First we have to calculate the AMAT of the L2:
   $AMAT_{L2} = 5ns + 0.3 * 30ns = 14ns = 14cc$ Now it turns out $AMAT_{L2}$ is the new miss penalty of the L1 caches.
   $AMAT_{I\$} = 1ns + 0.007 * 14ns = 1.098ns$
   $AMAT_{D\$} = 1ns + 0.06 * 15ns = 1.84ns$

   (d) What would the CPI of your processor be with the L2 cache? What is the speedup due to the L2 cache?

   $CPI = 1.2cc/instr + 0.007 * 14cc/instr + 0.06 * 0.38 * 14cc/instr = 1.617cc/instr$
   $speedup = 2.094/1.617 = 1.30x$

   (e) Why do you suppose the miss rate of the L2 is so much higher than that of the L1?

   The miss rate of the L2 is much higher than that of the L1 because the L1 filters out a lot of the locality in the data access, essentially skimming the cream off the top so to speak.

   (f) Why is the miss rate of the instruction cache lower than the miss rate of the data cache?

   Instruction access has a lot more temporal and spacial locality than data access. This is because instruction access is very sequential and loops repeatedly over same instructions.